Министерство образования Российской Федерации
Пензенский государственный университет
Кафедра«Вычислительная техника»

Отчёт

по лабораторной работе№3

по курсу «Л и О А в ИЗ»

на тему«Динамические списки»

Выполнилистудентыгруппы24ВВВ4:
Суходолов И.А.
Чернышевский Е.И.

Приняли к.т.н.,доцент:

ЮроваО.В.

к.э.н.,доцент:

АкифьевИ.В.

Пенза2025

Цель: освоить практические навыки работы с динамическими структурами данных на языке С.

Общие сведения.

Список представляет собой последовательность элементов определенного типа. Простейший тип списка – линейный, когда для каждого из элементов, кроме последнего, имеется следующий, и для каждого, кроме первого имеется предыдущий элемент.

Возможна реализация списков посредством массивов или динамическая реализация.

Динамические списки относятся к динамическим структурам и используются, когда размер данных заранее неизвестен. Созданием динамических данных должна заниматься сама программа во время своего исполнения, этим достигается эффективное распределение памяти, но снижается эффективность доступа к элементам.

Динамические структуры данных отличаются от статических двумя основными свойствами:

1) в них нельзя обеспечить хранение в заголовке всей информации о структуре, поэтому каждый элемент должен содержать информацию, логически связывающую его с другими элементами структуры;
2) для них зачастую не удобно использовать единый массив смежных элементов памяти, поэтому необходимо предусматривать ту или иную схему динамического управления памятью.

Для обращения к динамическим данным применяют указатели.

Набор операций над списком будет включать добавление и удаление элементов, поиск элементов списка.

Различают односвязные, двусвязные и циклические списки.

В простейшем случае каждый элемент содержит всего одну ссылку на следующий элемент, такой список называется односвязным.

В простейшем случае для создания элемента списка используется структура, в которой объединяются полезная информация и ссылка на следующий элемент списка

Задание

1) Реализовать приоритетную очередь, путём добавления элемента в список в соответствии с приоритетом объекта (т.е. объект с большим приоритетом становится перед объектом с меньшим приоритетом).

2) *На основе приведенного кода реализуйте структуру данных *Очередь.*

3) *На основе приведенного кода реализуйте структуру данных *Стек.*

Листинг

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <locale.h>
#include <stdlib.h>
#include <string.h>

struct node
{
   char inf[256]; // полезная информация
   int priority;  // приоритет элемента (чем больше число, тем выше приоритет)
   struct node* next; // ссылка на следующий элемент
};

// Раздельные указатели для трех структур
struct node* head_queue = NULL, * last_queue = NULL;    // Очередь
struct node* head_stack = NULL, * last_stack = NULL;    // Стек
struct node* head_priority = NULL, * last_priority = NULL; // Приоритетная очередь
```

```
int dlinna = 0;
char find_el[256];

// Функции
struct node* get_struct(void);
void spstore(void);
void spstec(void);
void sppriority(void);
void review_queue(void);
void review_stack(void);
void review_priority(void);
void del_queue(char* name);
void del_stack(char* name);
void del_priority(char* name);
struct node* find_queue(char* name);
struct node* find_stack(char* name);
struct node* find_priority(char* name);
void del_head_queue(void);
void del_head_stack(void);
struct node* dequeue_priority(void);
struct node* peek_priority(void);

// Функция создания элемента с приоритетом
struct node* get_struct(void)
{
   struct node* p = NULL;
   char s[256];
   int priority;

   if ((p = (struct node*)malloc(sizeof(struct node))) == NULL)
   {
```

```
        printf("Ошибка при распределении памяти\n");
        exit(1);
    }

    int c;
    while ((c = getchar()) != '\n' && c != EOF);

    printf("Введите название объекта: ");
    fgets(s, sizeof(s), stdin);

    s[strcspn(s, "\n")] = 0;

    printf("Введите приоритет объекта (целое число): ");
    scanf("%d", &priority);

    if (strlen(s) == 0)
    {
        printf("Запись не была произведена\n");
        free(p);
        return NULL;
    }

    strcpy(p->inf, s);
    p->priority = priority;
    p->next = NULL;

    return p;
}

/* Добавление в обычную очередь (в конец) */
void spstore(void)
{
```

```
    struct node* p = NULL;

    p = get_struct();
    if (p == NULL)
    {
        return;
    }

    // Если очередь пуста
    if (head_queue == NULL)
    {
        head_queue = p;
        last_queue = p;
    }
    else
    {
        last_queue->next = p;
        last_queue = p;
    }
    printf("Элемент добавлен в конец очереди: %s (приоритет %d)\n", p->inf, p->priority);
}

/* Добавление в стек (в начало) */
void spstec(void)
{
    struct node* p = NULL;

    p = get_struct();
    if (p == NULL)
    {
        return;
```

```
    }

    // Если стек пуст
    if (head_stack == NULL)
    {
        head_stack = p;
        last_stack = p;
    }
    else
    {
        p->next = head_stack;
        head_stack = p;
    }
    printf("Элемент добавлен в стек: %s (приоритет %d)\n", p->inf, p->priority);
}

/* Добавление в приоритетную очередь (с сортировкой по приоритету) */
void sppriority(void)
{
    struct node* p = NULL;
    struct node* current = NULL;
    struct node* prev = NULL;

    p = get_struct();
    if (p == NULL)
    {
        return;
    }

    // Если приоритетная очередь пуста или новый элемент имеет высший приоритет
    if (head_priority == NULL || p->priority > head_priority->priority)
```

```
    {
        p->next = head_priority;
        head_priority = p;
        if (last_priority == NULL)
        {
            last_priority = p;
        }
    }
    else
    {
        // Ищем место для вставки
        current = head_priority;
        while (current != NULL && current->priority >= p->priority)
        {
            prev = current;
            current = current->next;
        }

        // Вставляем между prev и current
        prev->next = p;
        p->next = current;

        // Если вставляем в конец, обновляем last_priority
        if (current == NULL)
        {
            last_priority = p;
        }
    }
    printf("Элемент добавлен в приоритетную очередь: %s (приоритет %d)\n", p->inf, p->priority);
}
```

```
/* Просмотр обычной очереди */
void review_queue(void)
{
   struct node* struc = head_queue;
   if (head_queue == NULL)
   {
      printf("Очередь пуста\n");
      return;
   }

   printf("Содержимое очереди:\n");
   while (struc)
   {
      printf("Имя - %s, Приоритет - %d\n", struc->inf, struc->priority);
      struc = struc->next;
   }
}

/* Просмотр стека */
void review_stack(void)
{
   struct node* struc = head_stack;
   if (head_stack == NULL)
   {
      printf("Стек пуст\n");
      return;
   }

   printf("Содержимое стека:\n");
   while (struc)
   {
      printf("Имя - %s, Приоритет - %d\n", struc->inf, struc->priority);
```

```
        struc = struc->next;
    }
}

/* Просмотр приоритетной очереди */
void review_priority(void)
{
    struct node* struc = head_priority;
    if (head_priority == NULL)
    {
        printf("Приоритетная очередь пуста\n");
        return;
    }

    printf("Содержимое приоритетной очереди:\n");
    while (struc)
    {
        printf("Имя - %s, Приоритет - %d\n", struc->inf, struc->priority);
        struc = struc->next;
    }
}

/* Поиск в обычной очереди */
struct node* find_queue(char* name)
{
    struct node* struc = head_queue;
    if (head_queue == NULL)
    {
        printf("Очередь пуста\n");
        return NULL;
    }
```

```
    while (struc)
    {
        if (strcmp(name, struc->inf) == 0)
        {
            return struc;
        }
        struc = struc->next;
    }

    printf("Элемент не найден в очереди\n");
    return NULL;
}

/* Поиск в стеке */
struct node* find_stack(char* name)
{
    struct node* struc = head_stack;
    if (head_stack == NULL)
    {
        printf("Стек пуст\n");
        return NULL;
    }

    while (struc)
    {
        if (strcmp(name, struc->inf) == 0)
        {
            return struc;
        }
        struc = struc->next;
    }
```

```
    printf("Элемент не найден в стеке\n");
    return NULL;
}

/* Поиск в приоритетной очереди */
struct node* find_priority(char* name)
{
    struct node* struc = head_priority;
    if (head_priority == NULL)
    {
        printf("Приоритетная очередь пуста\n");
        return NULL;
    }

    while (struc)
    {
        if (strcmp(name, struc->inf) == 0)
        {
            return struc;
        }
        struc = struc->next;
    }

    printf("Элемент не найден в приоритетной очереди\n");
    return NULL;
}

/* Удаление из обычной очереди по имени */
void del_queue(char* name)
{
    struct node* struc = head_queue;
    struct node* prev = NULL;
```

```
int flag = 0;

if (head_queue == NULL)
{
   printf("Очередь пуста\n");
   return;
}

if (strcmp(name, struc->inf) == 0)
{
   flag = 1;
   head_queue = struc->next;
   if (head_queue == NULL)
   {
      last_queue = NULL;
   }
   free(struc);
   printf("Элемент удален из очереди\n");
   return;
}
else
{
   prev = struc;
   struc = struc->next;
}

while (struc)
{
   if (strcmp(name, struc->inf) == 0)
   {
      flag = 1;
      prev->next = struc->next;
```

```
            if (struc == last_queue)
            {
               last_queue = prev;
            }
            free(struc);
            printf("Элемент удален из очереди\n");
            return;
         }
         prev = struc;
         struc = struc->next;
      }

      if (flag == 0)
      {
         printf("Элемент не найден в очереди\n");
      }
}

/* Удаление из стека по имени */
void del_stack(char* name)
{
   struct node* struc = head_stack;
   struct node* prev = NULL;
   int flag = 0;

   if (head_stack == NULL)
   {
      printf("Стек пуст\n");
      return;
   }

   if (strcmp(name, struc->inf) == 0)
```

```
{
   flag = 1;
   head_stack = struc->next;
   if (head_stack == NULL)
   {
      last_stack = NULL;
   }
   free(struc);
   printf("Элемент удален из стека\n");
   return;
}
else
{
   prev = struc;
   struc = struc->next;
}

while (struc)
{
   if (strcmp(name, struc->inf) == 0)
   {
      flag = 1;
      prev->next = struc->next;
      if (struc == last_stack)
      {
         last_stack = prev;
      }
      free(struc);
      printf("Элемент удален из стека\n");
      return;
   }
   prev = struc;
```

```
        struc = struc->next;
    }

    if (flag == 0)
    {
        printf("Элемент не найден в стеке\n");
    }
}

/* Удаление из приоритетной очереди по имени */
void del_priority(char* name)
{
    struct node* struc = head_priority;
    struct node* prev = NULL;
    int flag = 0;

    if (head_priority == NULL)
    {
        printf("Приоритетная очередь пуста\n");
        return;
    }

    if (strcmp(name, struc->inf) == 0)
    {
        flag = 1;
        head_priority = struc->next;
        if (head_priority == NULL)
        {
            last_priority = NULL;
        }
        free(struc);
        printf("Элемент удален из приоритетной очереди\n");
```

```
            return;
        }
        else
        {
            prev = struc;
            struc = struc->next;
        }

        while (struc)
        {
            if (strcmp(name, struc->inf) == 0)
            {
                flag = 1;
                prev->next = struc->next;
                if (struc == last_priority)
                {
                    last_priority = prev;
                }
                free(struc);
                printf("Элемент удален из приоритетной очереди\n");
                return;
            }
            prev = struc;
            struc = struc->next;
        }

        if (flag == 0)
        {
            printf("Элемент не найден в приоритетной очереди\n");
        }
    }
```

```
/* Удаление первого элемента из обычной очереди */
void del_head_queue(void)
{
   if (head_queue == NULL)
   {
      printf("Очередь пуста\n");
      return;
   }

   struct node* temp = head_queue;
   head_queue = head_queue->next;

   if (head_queue == NULL)
   {
      last_queue = NULL;
   }

   printf("Удален элемент из очереди: %s (приоритет %d)\n", temp->inf,
temp->priority);
   free(temp);
}

/* Удаление первого элемента из стека */
void del_head_stack(void)
{
   if (head_stack == NULL)
   {
      printf("Стек пуст\n");
      return;
   }

   struct node* temp = head_stack;
```

```
    head_stack = head_stack->next;

    if (head_stack == NULL)
    {
        last_stack = NULL;
    }

    printf("Удален элемент из стека: %s (приоритет %d)\n", temp->inf,
temp->priority);
    free(temp);
}

/* Показать элемент с наивысшим приоритетом (без удаления) */
struct node* peek_priority(void)
{
    return head_priority; // В приоритетной очереди первый элемент всегда с
наивысшим приоритетом
}

/* Извлечение элемента с наивысшим приоритетом (с удалением) */
struct node* dequeue_priority(void)
{
    struct node* temp = head_priority;

    if (head_priority == NULL)
    {
        printf("Приоритетная очередь пуста\n");
        return NULL;
    }

    head_priority = head_priority->next;
    if (head_priority == NULL)
    {
```

```
        last_priority = NULL;
    }

    temp->next = NULL;
    return temp;
}

/* Очистка всех структур */
void cleanup_all(void)
{
    struct node* temp;

    // Очистка очереди
    while (head_queue != NULL)
    {
        temp = head_queue;
        head_queue = head_queue->next;
        free(temp);
    }
    last_queue = NULL;

    // Очистка стека
    while (head_stack != NULL)
    {
        temp = head_stack;
        head_stack = head_stack->next;
        free(temp);
    }
    last_stack = NULL;

    // Очистка приоритетной очереди
    while (head_priority != NULL)
```

```
    {
        temp = head_priority;
        head_priority = head_priority->next;
        free(temp);
    }
    last_priority = NULL;
}

int main()
{
    setlocale(LC_ALL, "rus");

    int choice;
    char name[256];
    struct node* temp;

    while (1)
    {
        printf("\n=== МЕНЮ ===\n");
        printf("1. Добавить элемент в обычную очередь\n");
        printf("2. Добавить элемент в стек\n");
        printf("3. Добавить элемент в приоритетную очередь\n");
        printf("4. Просмотреть обычную очередь\n");
        printf("5. Просмотреть стек\n");
        printf("6. Просмотреть приоритетную очередь\n");
        printf("7. Найти элемент в обычной очереди\n");
        printf("8. Найти элемент в стеке\n");
        printf("9. Найти элемент в приоритетной очереди\n");
        printf("10. Удалить элемент из обычной очереди по имени\n");
        printf("11. Удалить элемент из стека по имени\n");
        printf("12. Удалить элемент из приоритетной очереди по имени\n");
        printf("13. Удалить первый элемент из обычной очереди\n");
```

```
printf("14. Удалить первый элемент из стека\n");
printf("17. Выход\n");
printf("Выберите действие: ");
scanf("%d", &choice);

switch (choice)
{
case 1:
   spstore();
   break;
case 2:
   spstec();
   break;
case 3:
   sppriority();
   break;
case 4:
   review_queue();
   break;
case 5:
   review_stack();
   break;
case 6:
   review_priority();
   break;
case 7:
   printf("Введите имя для поиска в очереди: ");
   scanf("%s", name);
   temp = find_queue(name);
   if (temp != NULL)
   {
      printf("Найден в очереди: %s с приоритетом %d\n", temp->inf,
```

```
temp->priority);
            }
            break;
        case 8:
            printf("Введите имя для поиска в стеке: ");
            scanf("%s", name);
            temp = find_stack(name);
            if (temp != NULL)
            {
                printf("Найден в стеке: %s с приоритетом %d\n", temp->inf,
temp->priority);
            }
            break;
        case 9:
            printf("Введите имя для поиска в приоритетной очереди: ");
            scanf("%s", name);
            temp = find_priority(name);
            if (temp != NULL)
            {
                printf("Найден в приоритетной очереди: %s с приоритетом
%d\n", temp->inf, temp->priority);
            }
            break;
        case 10:
            printf("Введите имя для удаления из очереди: ");
            scanf("%s", name);
            del_queue(name);
            break;
        case 11:
            printf("Введите имя для удаления из стека: ");
            scanf("%s", name);
            del_stack(name);
            break;
```

```
        case 12:
            printf("Введите имя для удаления из приоритетной очереди: ");
            scanf("%s", name);
            del_priority(name);
            break;
        case 13:
            del_head_queue();
            break;
        case 14:
            del_head_stack();
            break;
        case 15:
            temp = peek_priority();
            if (temp != NULL)
            {
                printf("Элемент с наивысшим приоритетом: %s (приоритет:
%d)\n",
                    temp->inf, temp->priority);
            }
            break;
        case 16:
            temp = dequeue_priority();
            if (temp != NULL)
            {
                printf("Извлечен элемент из приоритетной очереди: %s
(приоритет: %d)\n",
                    temp->inf, temp->priority);
                free(temp);
            }
            break;
        case 17:
            cleanup_all();
            printf("Память очищена. Выход...\n");
```

```
            exit(0);
        default:
            printf("Неверный выбор\n");
        }
    }

    return 0;
}
```

Результат работы программы

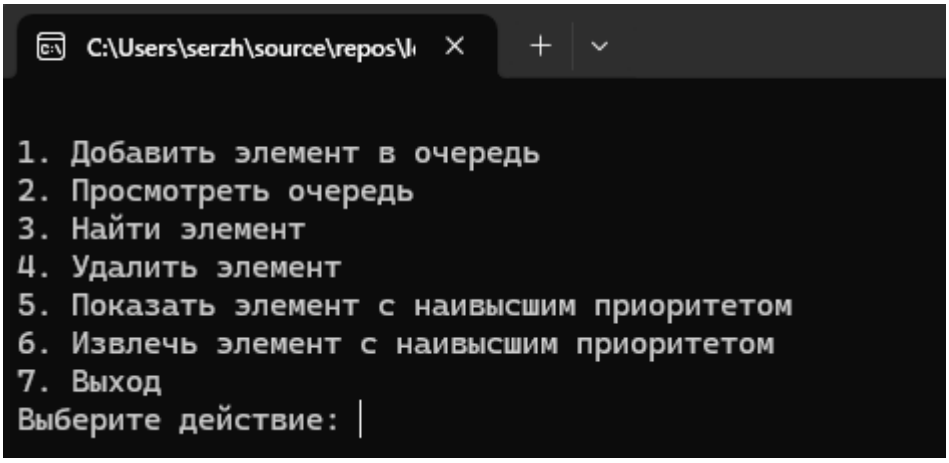

Рисунок1–Начальное меню

Вывод: в ходе выполнения данной лабораторной работы освоили практические навыки работы с динамическими структурами данных на языке С. Была реализована приоритетная очередь на основе односвязного списка.